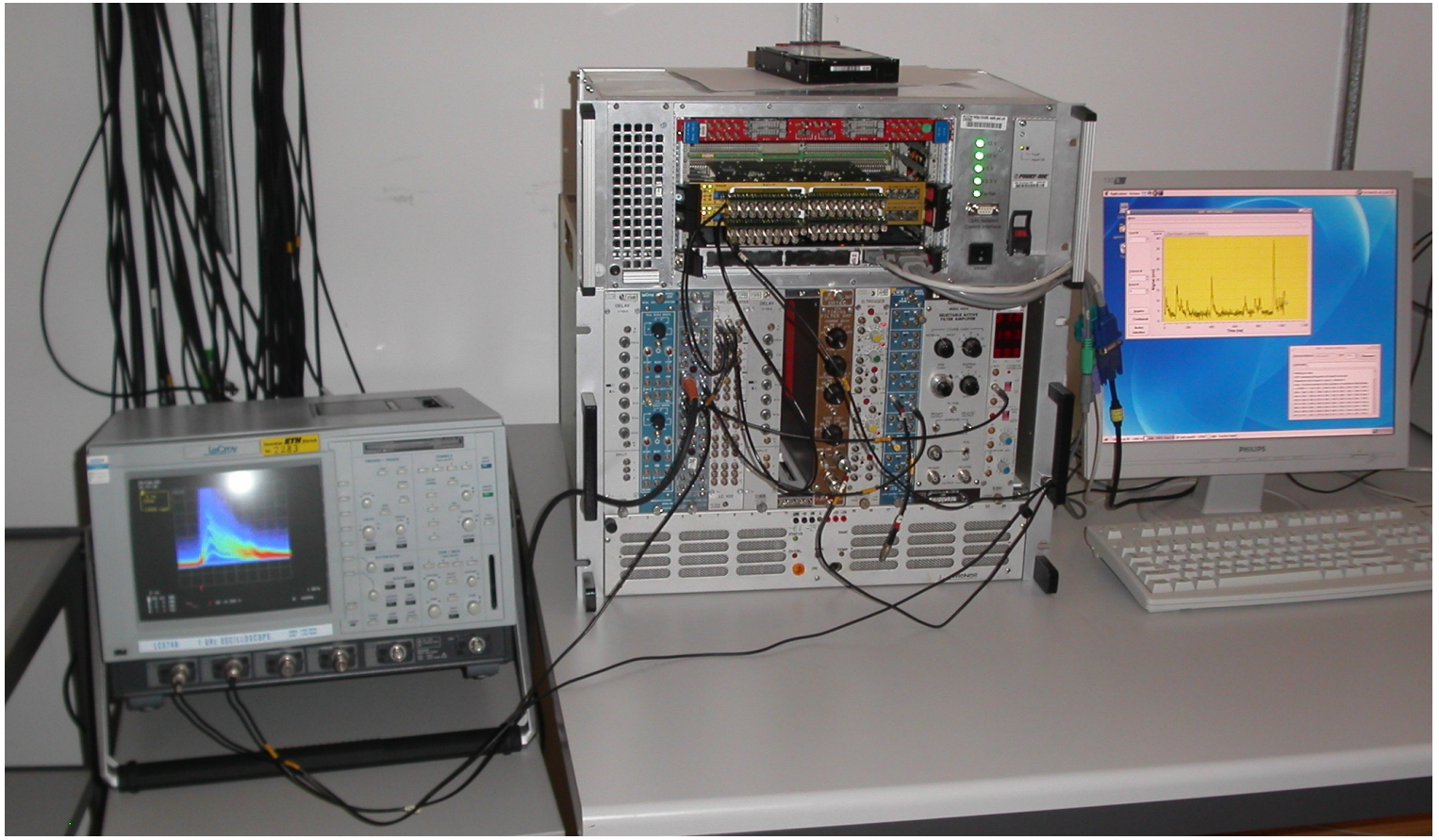
CT3/DWARF/FACT DAQ Status

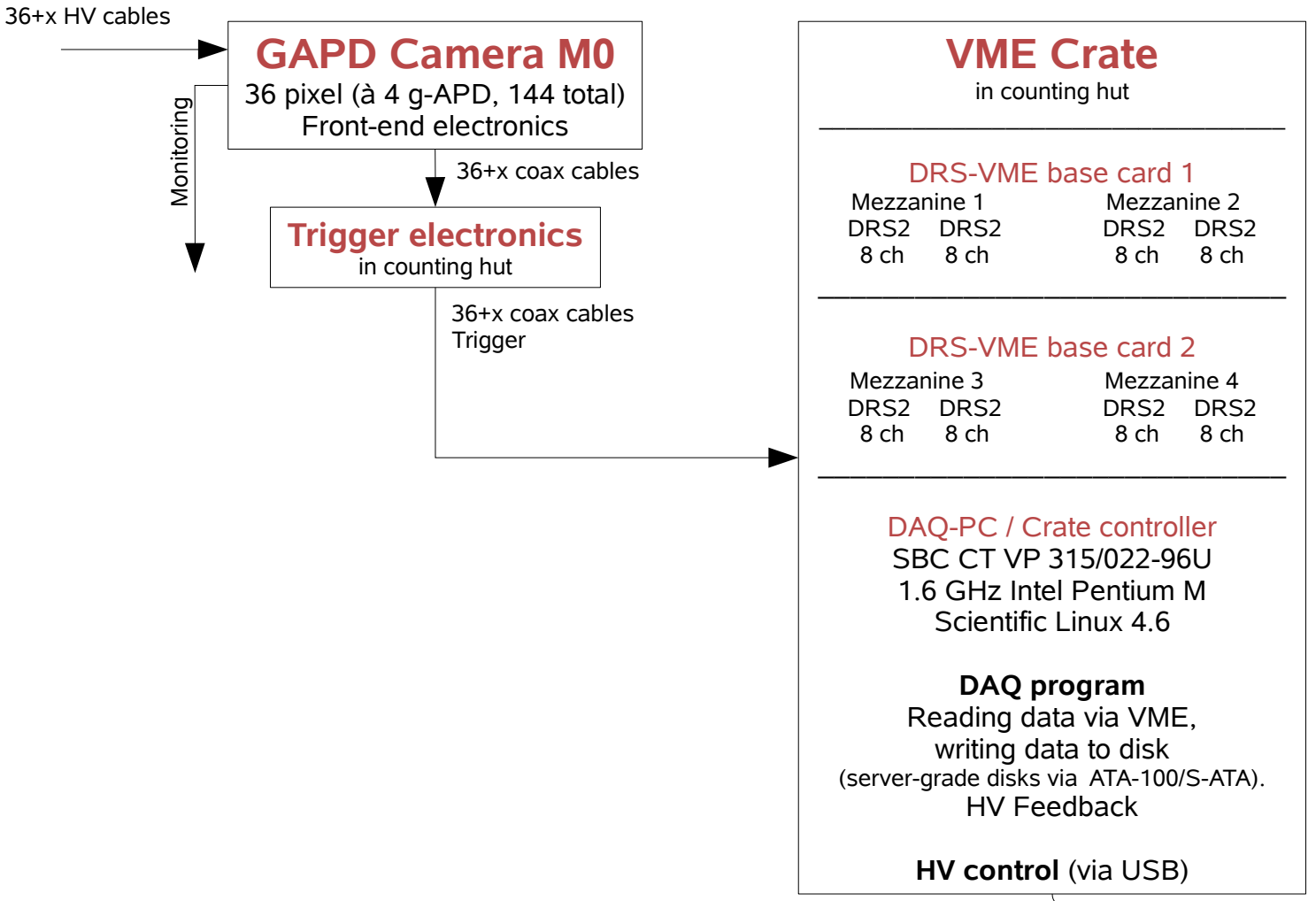


Oliver Grimm, ETH Zürich

3 March 2009

DAQ Chain for M0

DRS2-based



per trigger and channel:
2x1024 Byte = 2 kByte

48 channels (3 Mezzanine):
96 kByte/trigger

Trigger 100 Hz: **9.6 Mbyte/s**

If discarding ≈90% baseline data:
500 GByte in 144 hours

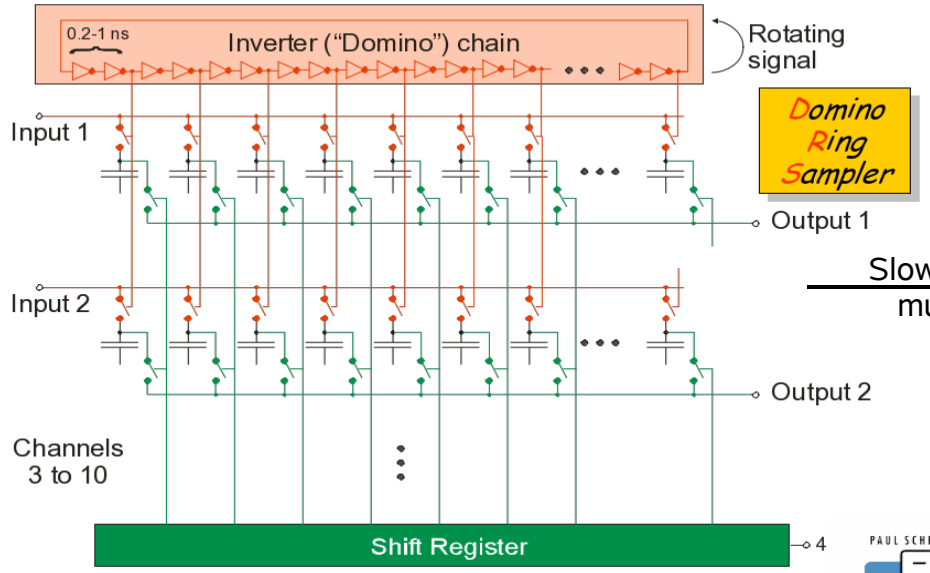
GZIP compression reduces to 30%,
but is slow

Measured achievable rate: 60-70 Hz (≈14 MByte/s)
- slight dependence on how much data written to disk

Permanent storage
(change external data disk or via ethernet)

DRS basics

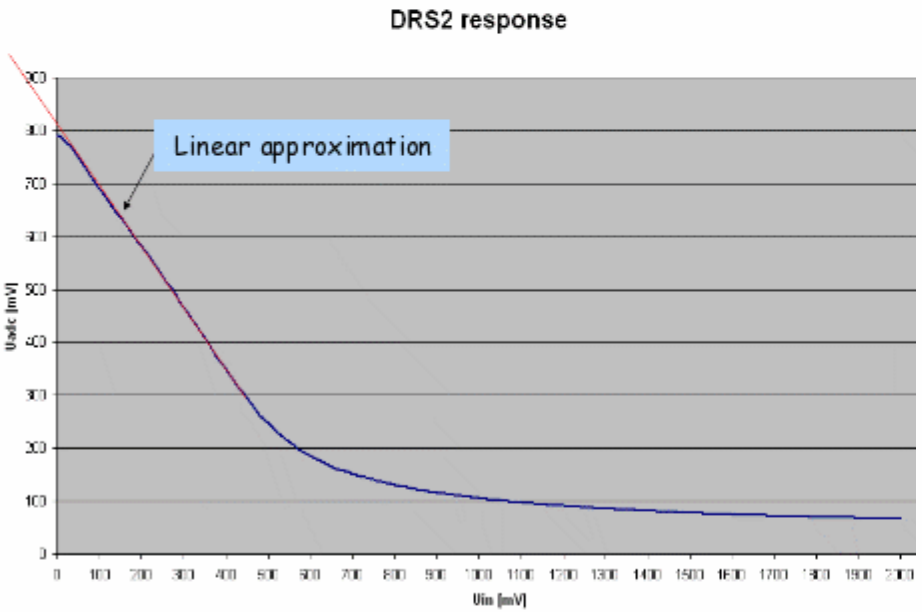
The DRS chip: principle of operation



Slow FADC via multiplexer

DRS3

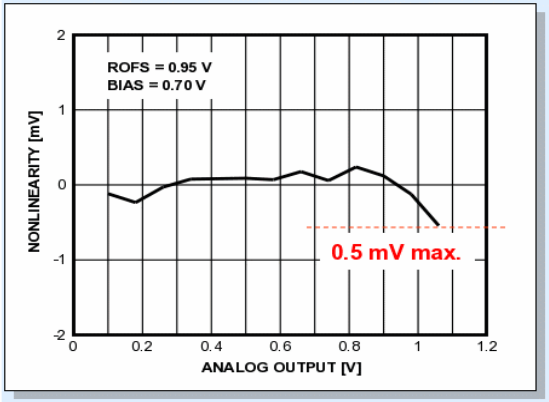
19 Oct. '04 IEEE/NSS Rome 2004

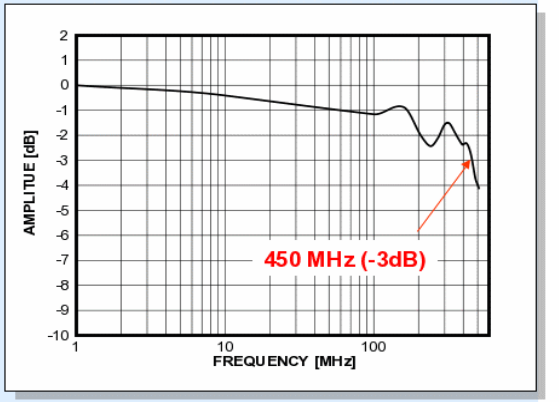


Bandwidth + Linearity

Readout chain shows excellent linearity from 0.1V … 1.1V @ 33 MHz reaout

Analog Bandwidth is currently limited by high resistance of on-chip signal bus, will be increased significantly with DRS4

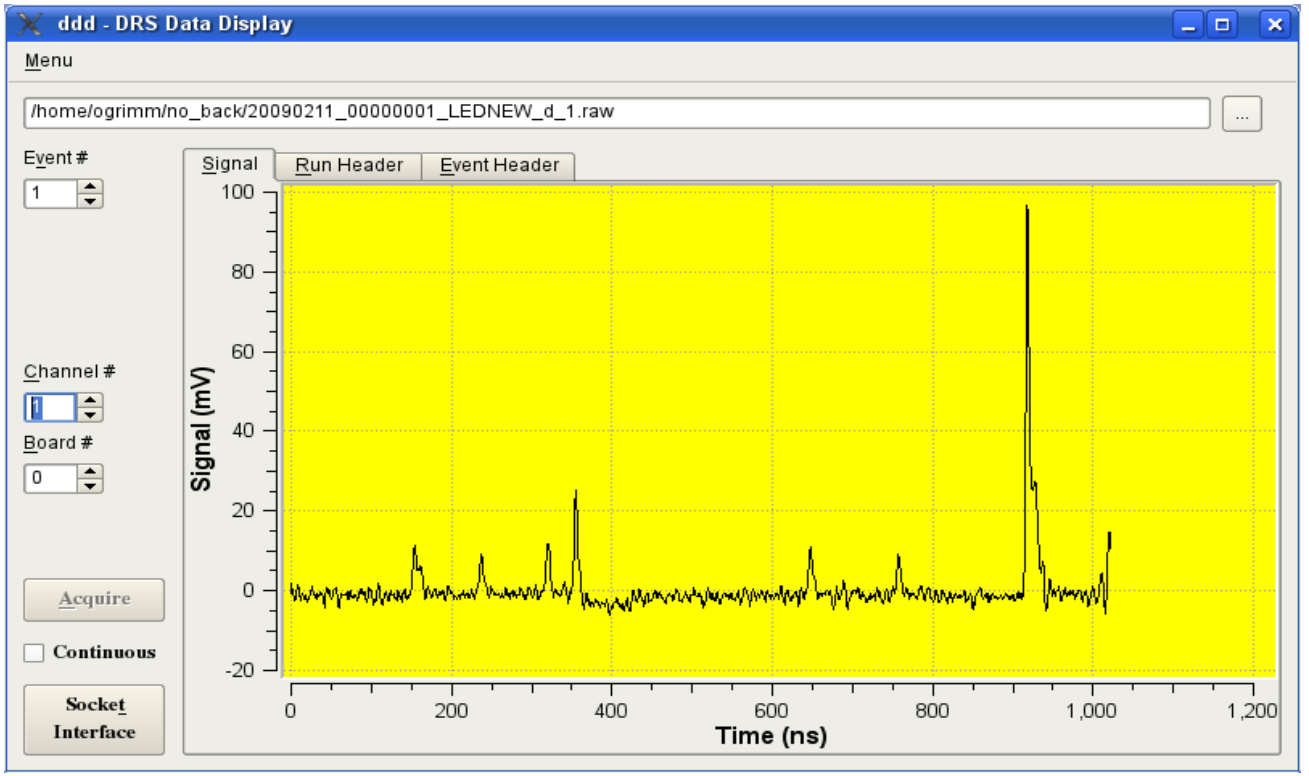




2 Nov. '07 IEEE/NSS Honolulu 2007 15

Slides courtesy Stefan Ritt/PSI

DRS2 Sampled signals



LED pulser

Sine generator

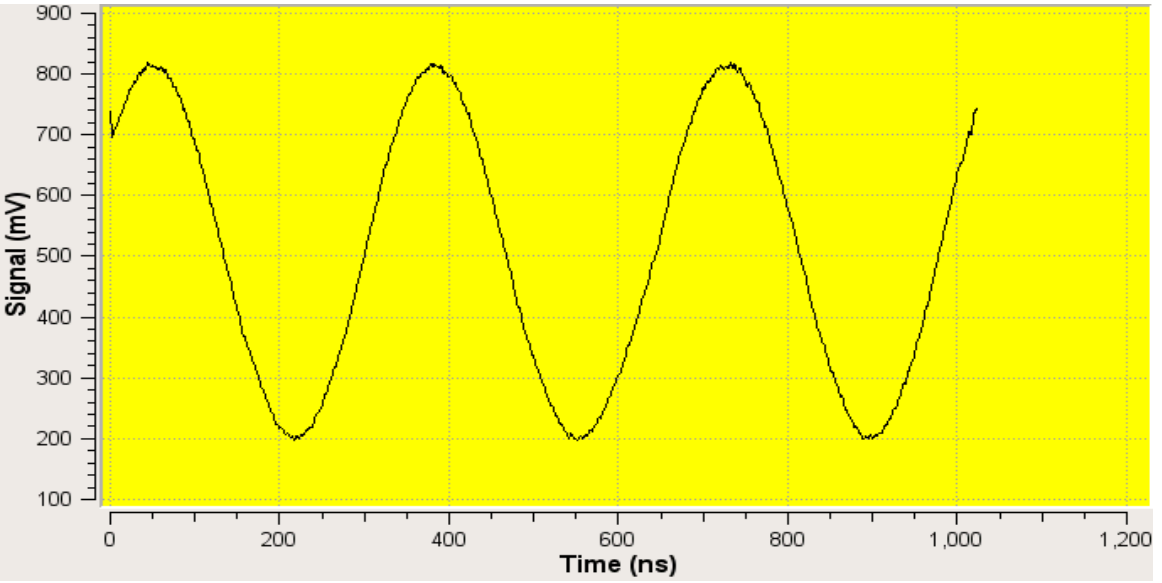
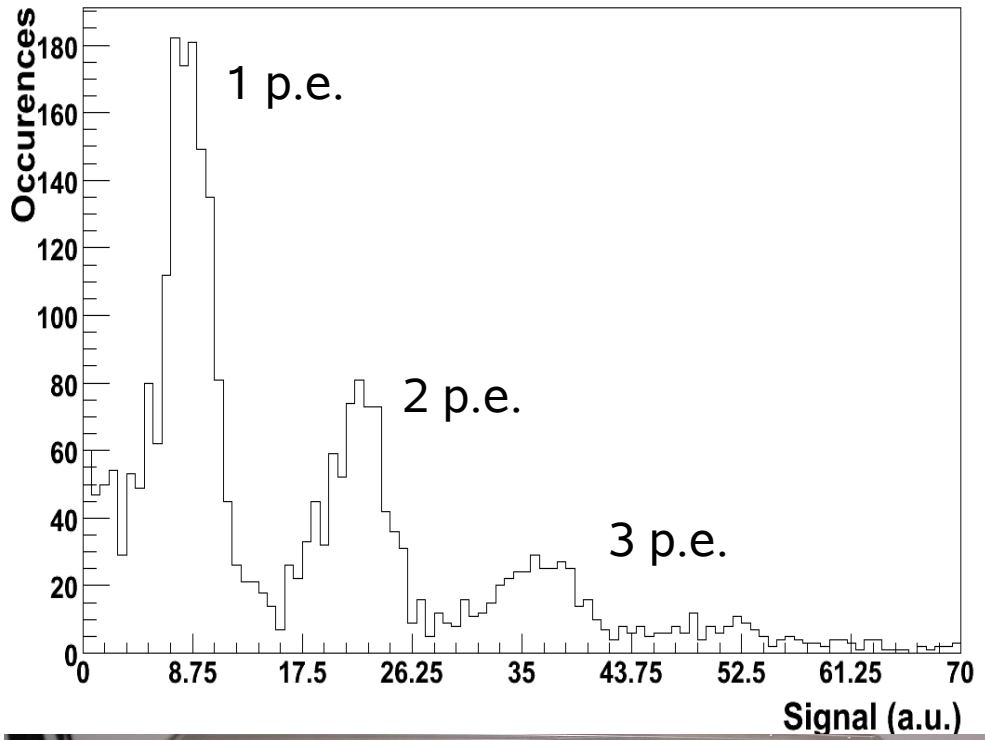
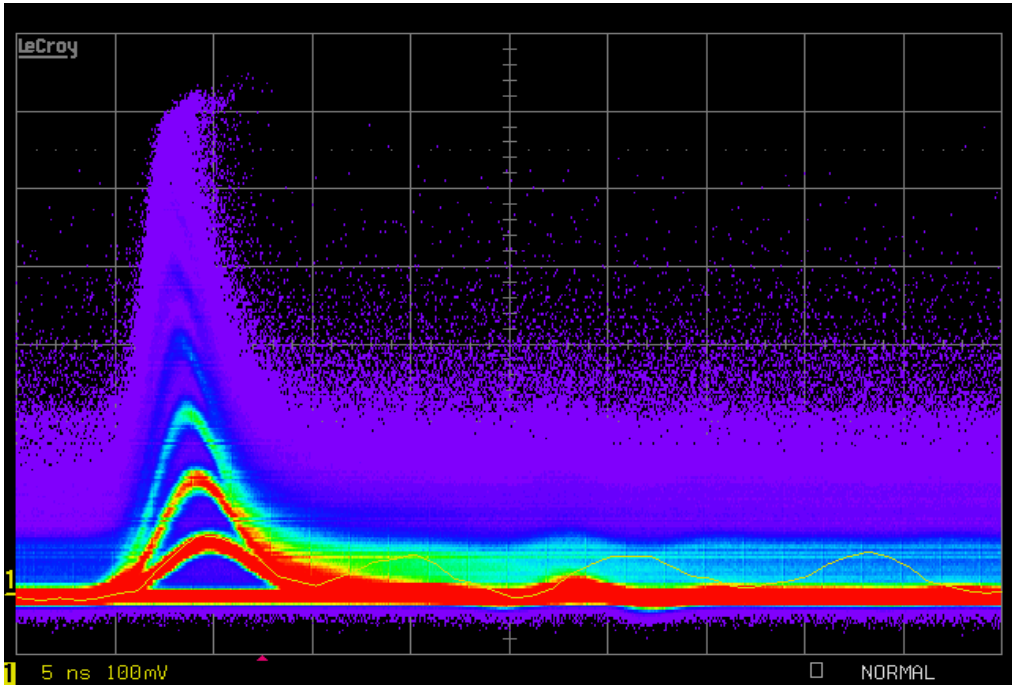
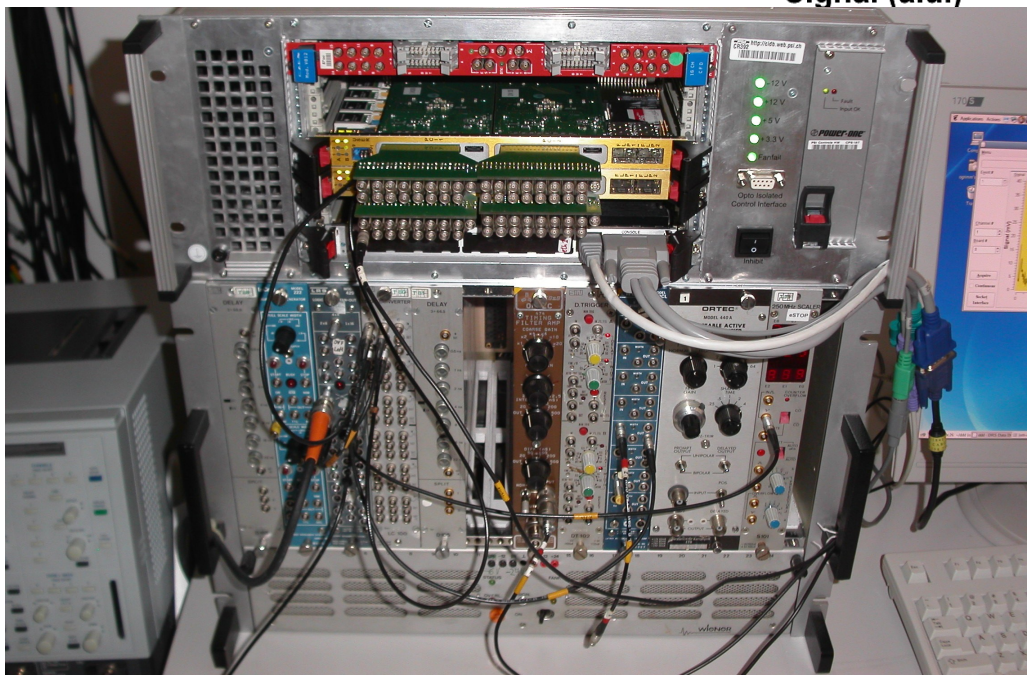
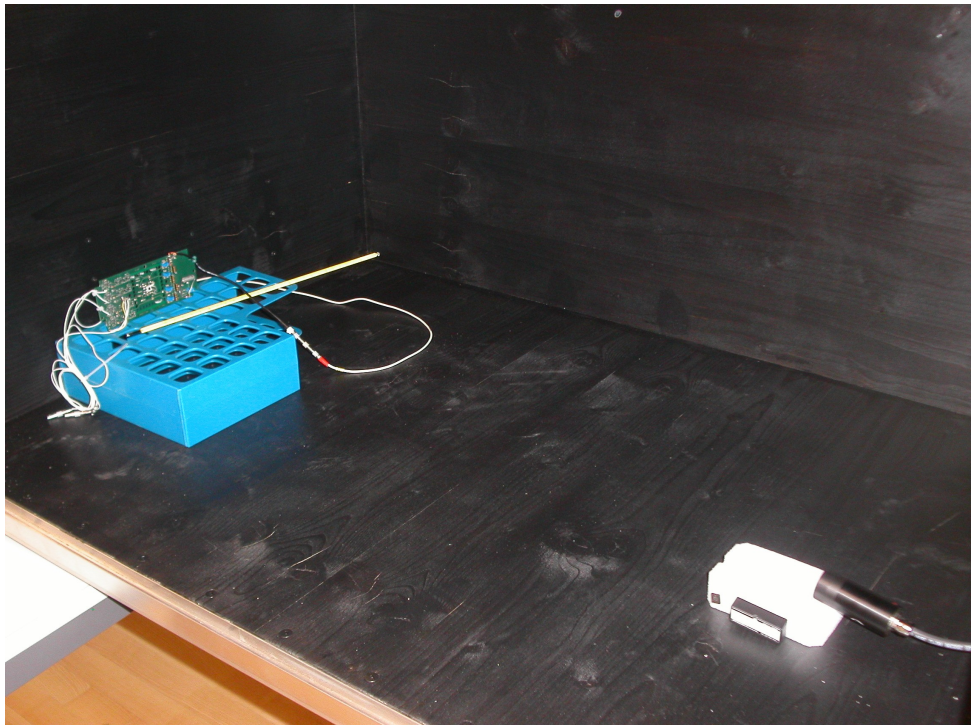


Photo-electron spectrum

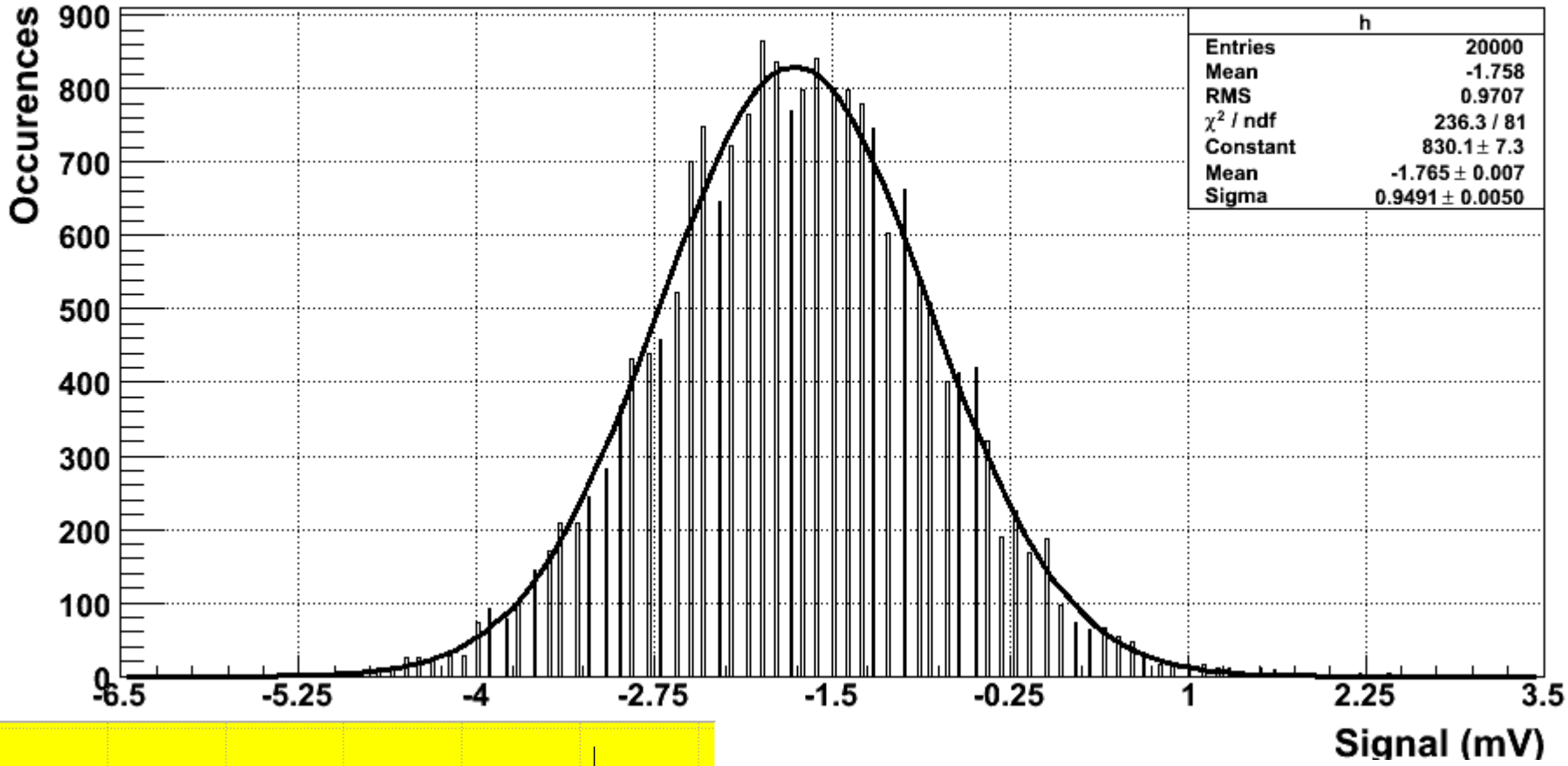


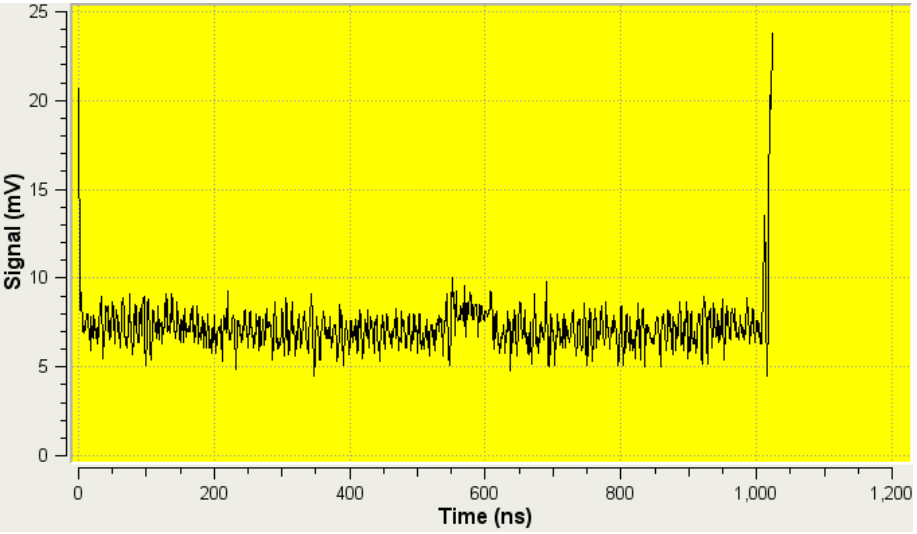






Baseline DRS2

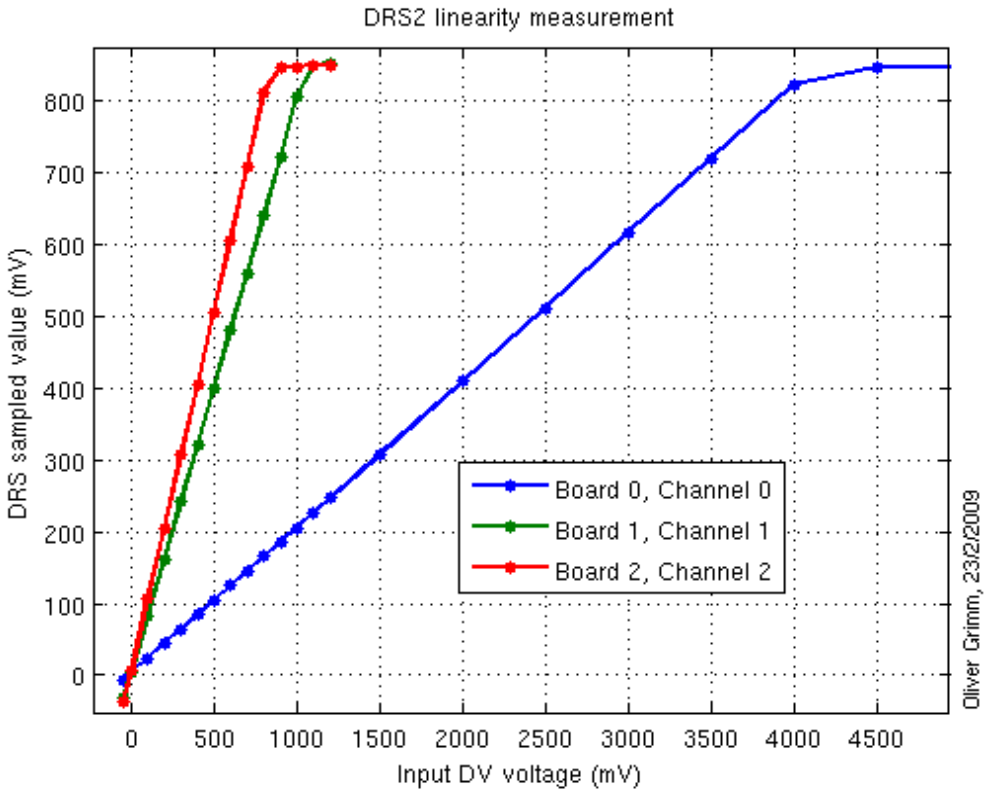


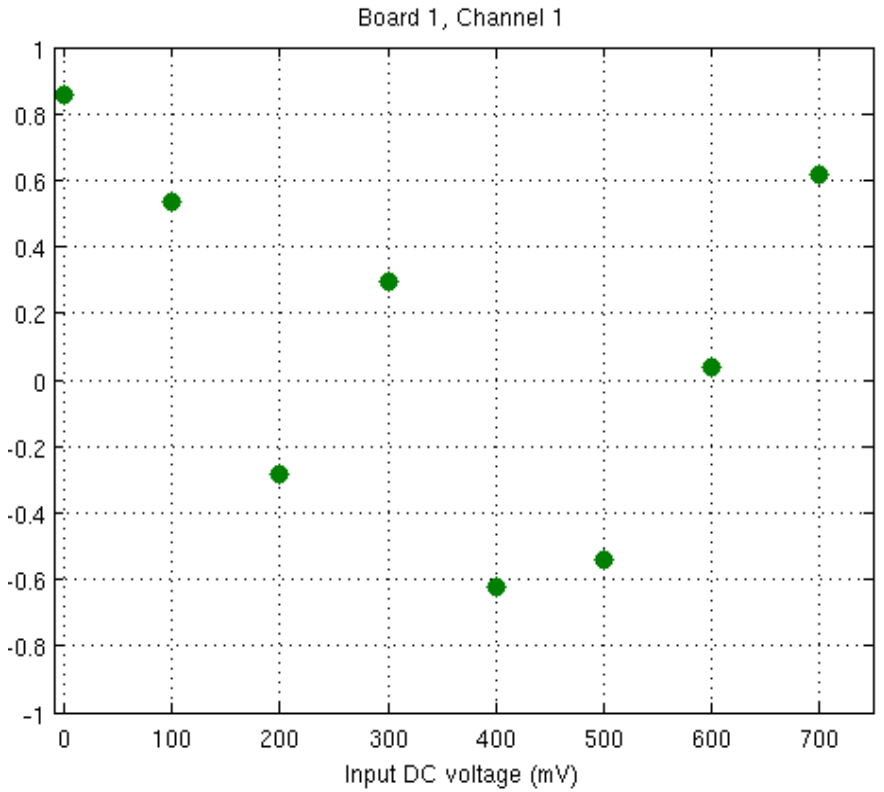


Measurement at ETH – for 1 V range
dynamic range 500:1 (9 bit)

According to S. Ritt/PSI:
DRS2 resolution 2 mV (rms)
DRS3/4: 350 µV/11.5 bit (single bin?)

DRS2 linearity



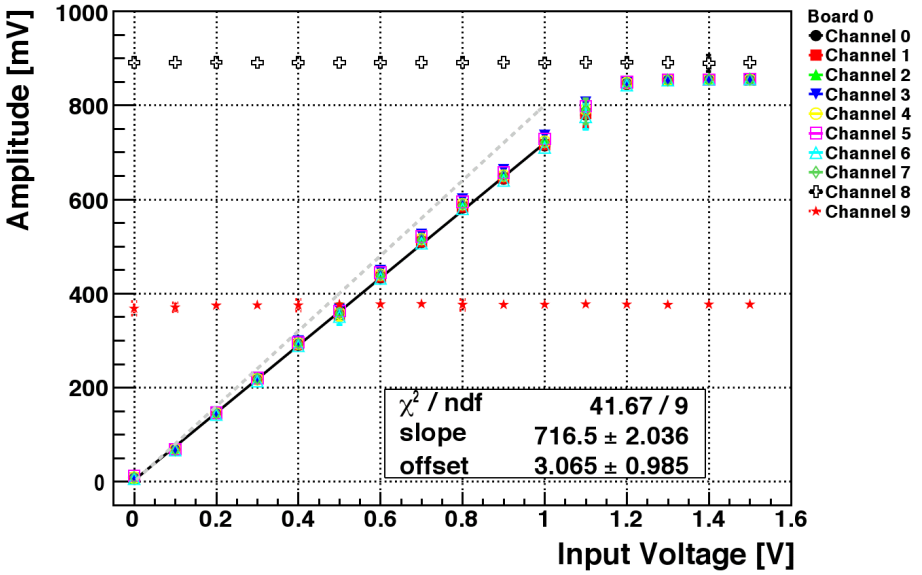


σ for all points (except at limits) 1-2 mV

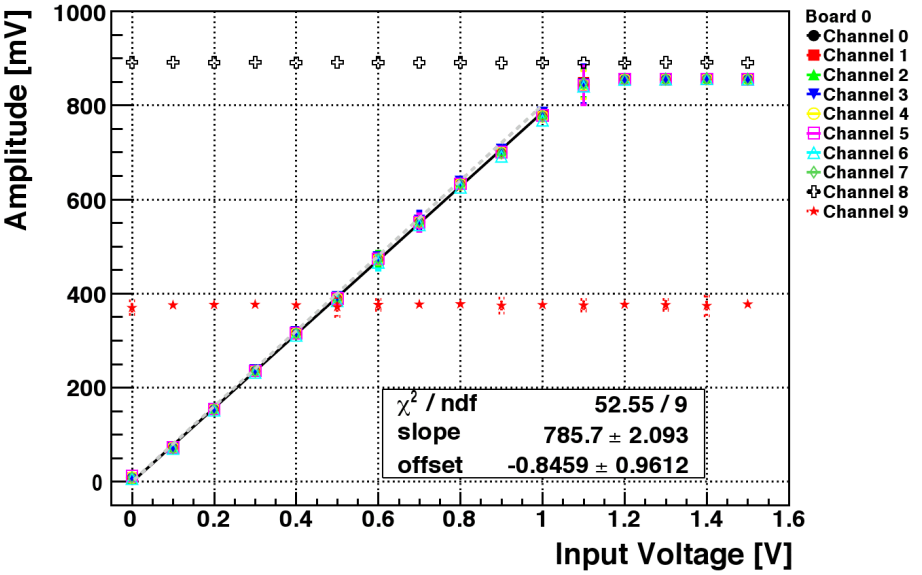
S. Ritt/PSI: “fully trust linearity only up to ≈500 mV”

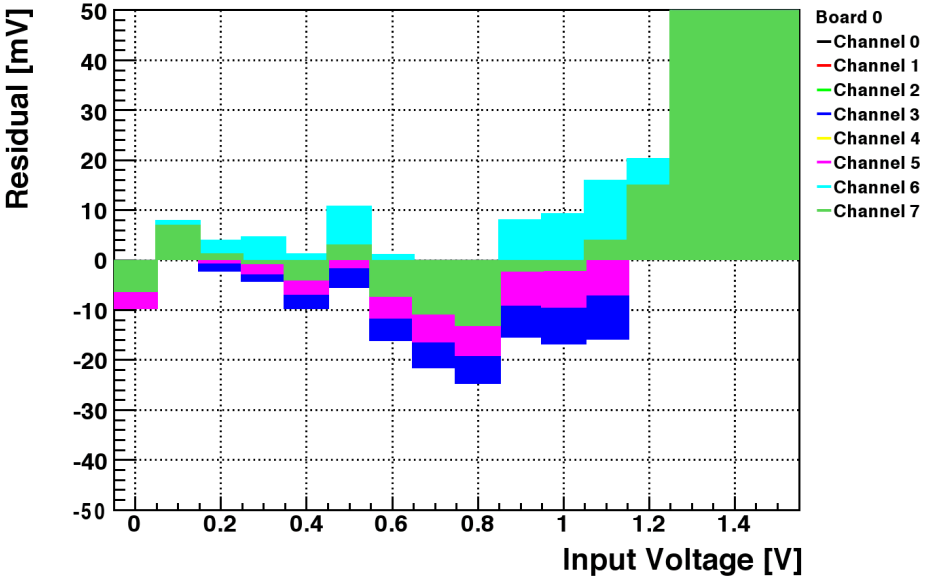
Linearity depends on pulse width

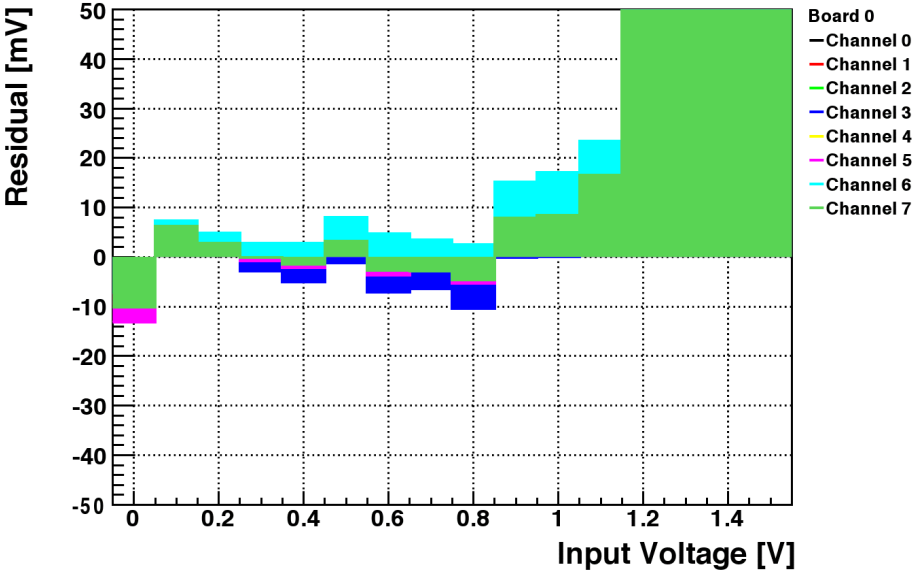
20 ns FWHM square wave



30 ns FWHM square wave







Slide courtesy S. Commichau

Why feedback needed?

Reasons for signal-amplitude drifts

1. *Gain* drifts directly with temperature (T up, gain down)
   - basic physical effect, nothing to do about it

2. *Gain* drifts indirectly with changing background rate (rate up, gain down)
   - voltage drop with increasing current reduced GAPD voltage
   - could be compensated by direct voltage feedback, but we don't have
   - gain will fluctuate due to instantaneous current fluctuation (unless bandwidth limited)

3. *Signal* amplitude drifts with changing background rate (rate up, signal down)
   - due to increasing cell occupancy, even if temperature and voltage absolutely stable
   - signal will fluctuate due to fluctuating cell occupancy

All three drifts (but not fluctuations) compensated by high voltage feedback

- regulating on signal from stable (or monitored) pulsed light source
- gain (single photoelectron amplitude) will not necessarily be stable due to 3.

Gain variation might be undesirable → foresee feedback on single-electron signal

If max. trigger rate ≈100 Hz, LED events will be few Hz. Feedback time scale will depend on signal fluctuation and desired precision (minutes).

Current features of HV feedback

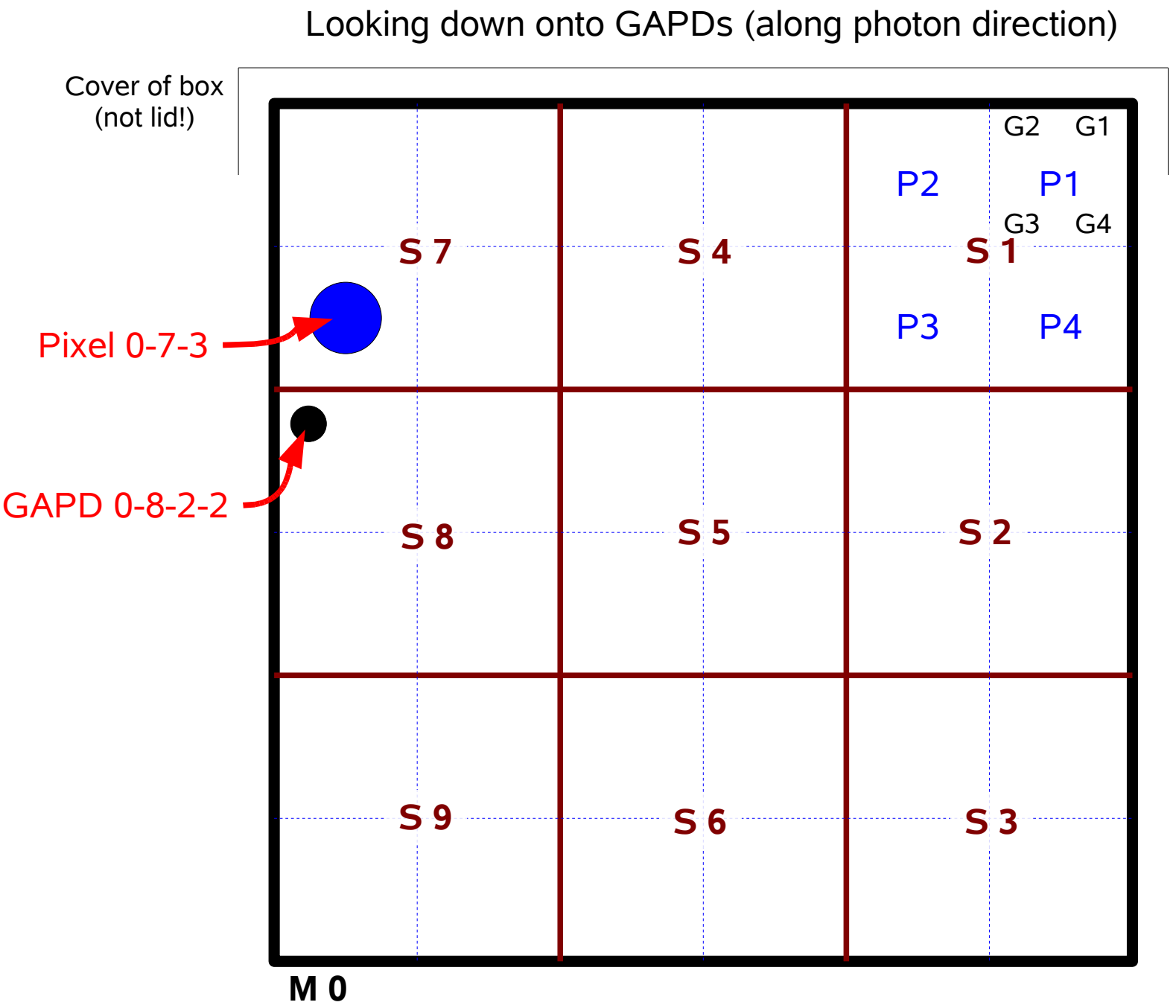
- Take average of current signal amplitude as target
- Set target manually
- Measure response matrix (assuming linear signal/voltage relationship)
- Feedback on target value
- Output of settings

...all coded but mostly untested.

High voltage feedback is class within DAQ program.
Communication to HV control program via TCP socket.

Identification of single-electron signals yet to be implemented.

Pixel/GAPD nomenclature



M: Module
S: Superpixel
P: Pixel
G: GAPD

Note:

Module #, DRS channel # and HV channel # start at zero.

Superpixel #, pixel # and GAPD # start at one.

Full label of geometric GAPD position: **mm-ss-pp-gg** (number of digits per field irrelevant)

mm	Module number
ss	Super-pixel number within module
pp	Pixel number within super-pixel
gg	GAPD number with pixel

Proposed format of `PixelMap.txt` requires fixed field order, but allows several field separators to facilitate human readability (space, tab and comma in arbitrary combination)
Lines starting with # considered as comments, empty lines allowed.

```
#Pixel#, DRS-Board# DRS-Chip# DRS-Channel#, HV-Chain# HV-Channel#
#-----------------------------------------------------------------

0-1-1,        0 0 0,    0 0
0-1-2,        0 0 1,    0 1
```

Few ideas for “central control” and GUI

Central control

- Text-based
- Clients connect via TCP socket
- Starts all required clients and checks for their well-being
- Acts as slow data collector
- Separate GUI is just another client
- Everything time stamped using time server and hardware clock
- Special handling of alarms/warnings via GUI

Presumably not available for M0 test.

Detailed work will be done on subsystems via their individual consoles/sockets.

GUI

- Contains few buttons to start subsystems, start run, set telescope, ...
- Health and alarm indicators
- Based on Qt graphical user interface (using root data displays instead of qwt?)
- Displays with low rate actual DAQ data (also send via socket)
- Event display?

Basic version might be availble for M0 test.

Data organization – Header file RawDataCTX.h

- With header file and shared library RawDataCTX.so data can be read from most programs
- Only little effort of being platform-independent (very little with the DAQ program itself)

Format not yet fixed, but should be frozen soon.

```
#define DATA_FORMAT 1

#define I8      char
#define U8      unsigned char
#define I16     short
#define U16     unsigned short
#define I32     int
#define U32     unsigned int
#define F32     float

#define MAGICNUM_FILE_OPEN 0xe0e1    // Magic number for run header while file open
#define MAGICNUM_FILE_CLOSED 0xe0e0  //    ... and when file is closed

#define MAX_NUM_CMCBOARDS 10
```

```
typedef struct {  // Run header
  U16 MagicNum;
  I8  Name[5];           // "RUNH", NULL-terminated
  U16 DataFormat;        // Increasing whenever header format changes
  I8  DAQVersion[12];    // contains result of __DATE__ macro
  I8  Source[16];
  I8  Type;               // p=pedestal, d=data, t=test
  U32 RunNumber;

  U16 StartYear;
  U8  StartMonth;
  U8  StartDay;
  U8  StartHour;
  U8  StartMinute;
  U8  StartSecond;

  U16 EndYear;
  U8  EndMonth;
  U8  EndDay;
  U8  EndHour;
  U8  EndMinute;
  U8  EndSecond;     ...

                       U32 Events;                     // Number of events in the run
  F32 SourceRA;        U32 NCMCBoards;                 // Number of used boards
  F32 SourceDEC;       U32 NChips;                // Number of DRS chips per board
  F32 TelescopeRA;     U32 NChannels;         // Number of channels per chip
  F32 TelescopeDEC;    U32 Samples;                    // number of samples
                       I32 Offset;                     // offset from first sample
...
                       I32 SerialNo[MAX_NUM_CMCBOARDS];   // Board serial number
                       F32 NomFreq[MAX_NUM_CMCBOARDS];    // nominal sampling frequency [GHz]
                       F32 BoardTemp[MAX_NUM_CMCBOARDS];  // board temperature [C]
                     } __attribute__((__packed__)) RunHeader;
```

```
typedef struct { // Event header
  I8  Name[5];          // "EVTH", NULL-terminated
  U32 EventNumber;
  F32 TimeSec;          // event time stamp in seconds, ms precision
  U16 TriggerType;
} __attribute__((__packed__)) EventHeader;


int fWriteRunHeader(RunHeader *, FILE *);
int fReadRunHeader(RunHeader *, FILE *);
int PrintRunHeader(RunHeader *, FILE *);

int fWriteEventHeader(EventHeader *, FILE *);
int fReadEventHeader(EventHeader *, FILE *);
int PrintEventHeader(EventHeader *, FILE *);

int fWriteCMCData(float *, FILE *, RunHeader *, int);
int fReadCMCData(float *, FILE *, RunHeader *);
```

Data organization on disk

```
          Board 1      Board 2      ...     Board 1      Board 2      ...
   RH  EH C1 C2 C3 ... C1 C2 C3 ... ...  EH C1 C2 C3 ... C1 C2 C3 ... ...
       -------------------------------  -------------------------------
       Event 1                          Event 2
```

RH Run header EV Event header Cx Channel (0-19 for 2 chips)

Channel data are written as floats, length of channel data is in the run header

Filename convention

Pattern	`Date_RunNumber_Target_DataType_FilePart.raw`
Example	`20090217_00000001_DUMMY_p_0.raw`

Date	Run start date
Run number	Increased automatically with every start of DAQ - determined from alphabetically last file in directory
Target	Name of run target as given by user
Data type	Data, Pedestal, Test, .. run
File part	Filesize limited to 1 GByte, part number increased with every file

M0 DAQ component status

Main set-up

All hardware components in house, assembled and running. Some mechanical improvements needed.

Read-out software available and installed, only few tests so far.

Spare components

1 VME crate	will be brought from CERN by Werner
1 SBC CT VP 315/022-96U (with break-out box)	available (rented from CERN electronics pool)
2 server-grade hard disks	to be ordered
1 VME DRS base card	reserved at PS/S. Ritt
2 DRS2 mezzanine cards (with DRS2 chips)	--- " ---
1 LEMO-interface board	available
1 ATA/S-ATA converter	to be ordered

Software needs to be installed, hard disk images to be made as further backup.

Prospects for “3 deg camera”

Could use same VME-based DAQ design with DRS4 chips

Assume **7 modules** (16 mezzanine boards) and **100 Hz** trigger rate.

DRS4 allows region of interest. If reading 10% of pipeline --> ≈5 MByte/s. **500 GByte** in approx. **28 hours**.

Next steps

Evaluate *Concurrent Technologies VX 511/06x SBC*

- 2.26 GHz Intel Core 2 Duo, 2eSST (best CT currently (?) available, possibly support from CERN/Markus Joos)

If not available reasonably soon, fall back to Struck VME controller (with more powerful PC than now).

Prices:	32 DRS4 chips	≈ EUR 3400,- (à EUR 105,-)
	16 DRS4 Mezzanine boards	≈ EUR 8000,- (à EUR 500,-, available March 2009?)
	8 VME base boards[#]	≈ EUR 40000,- (à EUR 5000,-, PSI price EUR 1750,-)

[#]Boards are the same for DRS2 and DRS4

Next steps

Software

* Test full DAQ chain including HV feedback
* Investigate feedback on single-electron signal
* Plan to incorporate extra board structure to become independent of `MAX_NUM_CMCBOARDS`
* Brush up code for consistent and (as far as reasonably possible) comprehensive error checking
* Check into svn repository (also all other tools) and set up `daqct3` user
* Finish example root script to read in data
* See if raw data can be written with 2 bytes per sample (instead of 4 for float)

Hardware

* Finish backup set-up (including full system set-up from scratch)
* Investigate more powerful CT single-board computer or Struck VME-interface with more powerful PC for 3-deg camera
* Continue (limited) test with USB DRS4 evaluation board
* Test DRS4 VME-based readout as soon as available
* Generally only limited studies with DRS2 planned